IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 11, 2020, accepted March 30, 2020, date of publication April 2, 2020, date of current version April 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2985222

Automatic Keyword and Sentence-Based Text Summarization for Software Bug Reports

SHUBHRA GOYAL JINDAL, (Student Member, IEEE), AND ARVINDER KAUR
University School of Information and Communication Technology, Guru Gobind Singh Indraprastha University, New Delhi 110078, India
Corresponding author: Shubhra Goyal Jindal (Shubhra.phd@ipu.ac.in)

ABSTRACT Text Summarization is a process which efficiently retrieves the relevant information from documents. The objective of the proposed, unsupervised approach is to summarize bug reports (software artefacts) with complete content and diversified information. The proposed approach utilizes Rapid Automatic Keyword Extraction and term frequency-inverse document frequency method to extract meaningful keywords and key-phrases with a relevant score. For sentence extraction, fuzzy C-means clustering is used to extracts sentences having high degree of membership from each cluster above a set threshold value. A rule-engine is used for sentence selection. The rules are generated with the domain knowledge and based on the extracted information by the keywords and sentences selected by the clustering method. Cohesive and coherent summary is generated by the proposed method on apache bug reports. For redundancy removal and to re-rank generated summary, hierarchical clustering is presented to enrich the extracted summary. The proposed approach is evaluated on newly constructed Apache project Bug Report Corpus (APBRC) and existing Bug Report Corpus (BRC). The results are compared on the basis of performance metrics such as precision, recall, pyramid precision and F-score. The experimental results depict that our proposed approach attains significant improvement over other baseline approaches such as BRC and LRCA. It also attains significant improvement over existing state-of-art unsupervised approaches such as Hurried, centroid and others. It extracts significant keyword phrases and sentences from each cluster to achieve full coverage and coherent summary. The results evaluated on APBRC corpus attains an average value of 78.22%, 82.18%, 80.10% and 81.66% for precision, recall, f-score and pyramid precision respectively.

INDEX TERMS Text summarization, rapid automatic keyword extraction, fuzzy c-means, hierarchical clustering, bug reports, rule engine.

I. INTRODUCTION

In recent years, plenty of information is available on the internet from several domains. With huge amount of available data, it is an arduous and time-consuming task to read entire text documents and retrieve relevant information. To automatically attain relevant information in brief, text summarization is used. Generating accurate summary of a text document is a complex task and requires human intelligence to extract meaningful information from the text. Automatic text summarization has been used in several domains such as document summary [1], [2], essay or news summary [3], [4] and e-mail summarization [5], [6]. Apart from these, several software repositories such as Jira, Bugzilla manages numerous bug reports [7]–[9] with several open source projects. To oversee diverse number of bug reports, several automation tasks have been conducted such as detection of duplicate bug reports [10], [11], fixation of bugs [12], bug report triaging [13]–[15] and others. To accomplish these tasks, software testers and developers need to wade through entire bug reports having hundreds of sentences. A tester or developer needs to understand history of bug reports with specific domain knowledge as it is not a generic text summarization. In this research work, authors have focused on generation of a summary of bug reports which are the most valued artefacts of software project. It encloses several attributes such as title, one-line description, BugID, detailed description, comments made by several contributors and others. This information is useful for locating and fixing the bug in software project. To read and understand an entire bug report is a tedious task and therefore, bug summarization is an emerging field of research to help several developers to improve bug resolution process.

In literature, two categories of algorithms exist: abstractive summarization and extractive summarization. In abstractive summarization, semantic-representation, word-order and

The associate editor coordinating the review of this manuscript and approving it for publication was Hui Liu.

natural language of a text is modified with same contextual meaning. In this area, a major breakthrough is achieved through deep learning techniques. Several researchers have worked on Convolution neural network (CNN) [16], Recurrent neural network (RNN) [17], Reinforcement learning and Generative Adversarial networks (GAN) [18] with high accuracy as compared to other approaches used in literature. The major shortcoming of applying deep learning methods is the un-availability of training data as it is a supervised approach and standard golden summaries are not available in every domain. Whereas, in extractive summarization, sentences are extracted from the text document with same order and language to produce a condensed summary. In literature, several supervised [19]–[21] and unsupervised [22], [23] approaches have been proposed to summarize bug reports automatically. In supervised approach, one such research is carried out by Rastkar *et al.*, [19] in which a corpus of 36 bug reports from various open source projects was created named Bug Report Corpus (BRC). For each bug report, 24 features were calculated which comprises of four categories: Lexical, participant, length and structure. Logistic regression classifier was used which was trained on corpus of manually generated golden summaries by annotators. The results illustrate a low precision of 57%, recall of 35%, 40% f-score and 66% pyramid precision. To improve upon the results of [19], Jiang *et al.* proposed another approach PRST [20]. The authors consider 36 bug reports of BRC corpus [19] and its duplicate bug reports and constructed a new corpus named Modified Bug Report Corpus (MBRC). Page-rank algorithm is used to compute the textual similarity among sentences and then probability of each sentence was computed using logistic regression classifier. The results were merged and then sentences with high probability value were selected to form a summary. There was a slight improvement in precision and pyramid precision with same values of recall and f-score.

In contrast, an unsupervised approach, assigns a value based on some measure to each sentence and top ranked sentences are selected to form a summary. Lotufo *et al.* [23] proposed an unsupervised approach to generate summary by investigating how a long report is scanned by developers. In another work, Mani *et al.* [22] employed four approaches viz., Maximal Marginal Relevance (MMR), centroid, diverse rank and grasshopper. In previous researches [19], [20], keywords, lexical similarity and sentence weight features were used as feature set.

In this work, we focus on unsupervised approach and new method is constructed based on keyword-based features and sentence-based features to facilitate bug report summarization. For keyword-based feature extraction, two methods term frequency-inverse document frequency(tf-idf) and Rapid Automatic Keyword Extraction (RAKE) are used. In previous work, approaches for keyword extraction focus on corpus-oriented statistics [24], [25]. To avoid this drawback, documented oriented methods were used [26] which utilizes natural language processing to identify parts of speech in combination with other statistical, machine learning or supervised approaches. To avoid these drawbacks, RAKE is used which is language independent, domain independent and unsupervised in nature. It gives reasonable precision along with simplicity and computational efficiency [27]. For sentence-based feature extraction, Fuzzy C-means clustering has been used as compared to other features such as length, position, title word, thematic word and others as used in past work [19], [28], [29].

In this work, first, Bug reports from five different projects of Apache software foundation are extracted using a tool named Bug report collection system (BRCS) [30]. Among several extracted attributes, one-line description, long description and comments made by various contributors of 21 bug reports are collected to form a small sized corpus named APBRC (Apache Project Bug Report Corpus). The textual data of bug reports are segmented into sentences and are preprocessed using standard preprocessing steps.

Second, for feature extraction, first keyword extraction is done. The keywords extracted by tf-idf and RAKE are considered as keyword features. In RAKE each concept/word is assigned a score by calculating the degree of content word in a sentence. To compute the score of each keyword or keyword phrase, the sum is computed on the basis of content words in a text. As compared to tf-idf approach, RAKE extracts more complicated phrases from the text that implicit more meaning than individual words. For selecting the unigram words that is not taken by the RAKE, authors further enhance it with the tf-idf method.

For sentence-based features, Optimum number of clusters were identified by several methods such as Gap Statistics, K-means, Silhuoette, and with-in-squares. Based on optimum cluster, fuzzy C-means clustering is applied on the bug reports. Membership value of each sentence is computed and based on it, each sentence is assigned to a cluster with minimum Euclidean distance from the center of the cluster. From each cluster, sentences with high degree of membership are selected.

Third, after extraction of all the features, several rules are generated and based on these rules, sentences are selected to form an extractive summary. A compression ratio of 20% to the original bug report is achieved. The approach is evaluated using several performance metrics namely; precision, recall, f-score and pyramid precision and found better results as compared to state of art methods present in the literature.

Finally, after generating the summary, a short summary is also generated using hierarchical clustering. Dendrograms are generated and with the help of it, sentences are re-ranked and one sentence is selected from a pair of similar sentences, to generate a short summary with no redundant information.

The paper is organized as follows. In section II, the motivation of the study is presented. In section III, preliminary concepts used in proposed approach is presented followed by research methodology in section IV. The implementation of the proposed approach is illustrated in section V followed by results in section VI. The threats to validity are presented

in section VII. The related work is reviewed in section VIII. Finally, section IX concludes the paper.

II. MOTIVATION

Bug reports are the most valued artefact of software system. It consists of several attributes such as description, id, title and some predefined fields and numerous comments made by various developers. In earlier work, it was stated that 200 bug reports gathered from Mozilla open source project refers other 275 bug reports. To read and comprehend all the bug reports manually is arduous task for any tester or developer, as each bug report consists of hundreds of sentences and duplicate sentences. To reduce the time and effort of software testers, a novel approach is developed for summarization of bug reports.

Our aim is to design an innovative, novel automatic text summarization approach that discern the domain knowledge of bug reports and generate high quality bug summary. Summary consists of prime sentences, viz., description and code snippets having '{', '}', '{tmp field}', 'sql', '<', 'public static' and '='. As stated in literature [21], Tf-idf, unigram, bigram and centroid methods cannot capture the code snippets from the description and comments which constitutes the most significant part of bug summary. To achieve these code snippets, author has used RAKE method which captures all the code snippets from textual data of bug reports which has not been covered by any previous work [19]–[21].

Secondly, several optimization techniques have been applied to generate an extractive summary of text documents which uncovers numerical features such as similarity among sentences of text data. Although, for bug summarization, each bug report is unique and all the sentences are different. Therefore, in lieu of optimization technique, fuzzy clustering is enforced to apprehend the important sentences based on centroid method. Fuzzy clustering attains superior results as compared to other unsupervised techniques [22], [23]. Consolidating both keywords and sentences, a rule engine is designed to construct a final unsupervised extractive summary of bug reports. Further, to generate a short and concise summary, hierarchical clustering is applied which removed redundant sentences from the generated summary on the basis of dendograms and re-rank the sentences.

III. PRELIMINARY CONCEPTS

Various concepts used for summarization of bug reports are discussed in this section. It includes pre-processing of textual data of bug reports, Rapid automatic keyword extraction, fuzzy clustering and hierarchical clustering.

A. TEXT PRE-PROCESSING

To generate a summary, some pre-processing is required to the set of raw bug reports. Standard pre-processing steps are applied which includes segmentation, tokenization, removal of stop words, removal of punctuation and stemming.

Segmentation: In this process, each sentence separated by a delimiter is extracted separately from bug reports. All extracted sentences are then stored in order of the original bug report.

Tokenization: It is segmentation of sentences which breaks text into meaningful elements called as tokens, I.e. words, symbols and phrases.

Removal of stopwords: In this step, most commonly used words such as 'the', 'a', 'and', 'this' are removed from the textual data, which does not have semantic information.

Removal of punctuation: In this, punctuations and special characters like interrogation, exclamation are removed.

Stemming: It converts the words to their root form by removing suffix and prefix from the words. For instance, word 'presentation' is reduced to its root form 'present'.

After performing all these steps, text data is converted into a structural representation called as Document term matrix (DTM). It represents the frequency of each term present in a document.

B. RAPID AUTOMATIC KEYWORD EXTRACTION (RAKE)

Rake is a keyword extraction algorithm which is domain independent. It partitions the textual data into candidate keywords which are sequence of one or more content word that occur in a text. It extracts candidate keywords by analyzing the frequency of co-occurrence of these content words within a candidate keyword. For keyword extraction, Rake splits the textual data into an array of words. Then, this array of words is split into sequence of contiguous words separated by phrase delimiters and stop word position. The sequence of contiguous word is called as candidate keyword and each candidate keyword is assigned a same position in the text. A matrix of word co-occurrence is constructed which indicates the frequency of co-occurrence of each content word within a candidate keyword. Candidate words are also called as sequence of one or more content words (informative words) that occur in a text. After the identification of each candidate keyword, each keyword is assigned a score. The sum of the score of each content word is the total score of each candidate keyword [30], [31]. The process of assigning the score for every keyword is illustrated as follows:

- First, the frequency (freq) of each content word (CW) is calculated in a given textual document represented by freq(CW).
- After computing the frequency, degree of a word is calculated, represented by deg (CW). To compute the degree, total number of words that appear in candidate keywords consisting the content word is counted.
- At last, ratio of degree of content word to frequency of content word is computed, represented by

$$Deg(CW)/Freq(CW) \quad (1)$$

TABLE 1. Sentences in bug Id HDFS-7707.

1.Edit log corruption due to delayed block removal again. *2. Edit log corruption is seen again, even with the fix of HDFS- 6825.* *3.Prior to HDFS-6825 fix, if dirx is deleted recursively, an Op_CLOSE can get into edit log for the filey under dirx, thus corrupting the edit log restarting NN with the edit log would fail.*

TABLE 2. Score computation of various content words.

Content words	Edit	Log	Corrup -tion	delaye d	Block	Re mo v- al	Fi x
Deg(CW)	12	12	6	3	3	3	2
Freq(CW)	5	5	2	1	1	1	2
Ratio (Deg/fre q)	2.4	2.4	3	3	3	3	1
Content words	Recur - sively	Op_ clos e	Corrup -ting	Filey	Restart -ing	Dir x	fai l
Deg(CW)	2	1	1	1	1	2	1
Freq(CW)	1	1	1	1	1	2	1
Ratio (Deg/fre q)	2	1	1	1	1	1	1

Candidate Words:
1. edit-log-corruption
2.delayed block-removal
3.fix-dirx recursively
4.op_close – edit
5.log – filey – dirx
6.corrupting – edit log-
7.restarting NN – edit log- fail

The computation of score of candidate keywords is illustrated on text of bug Id HDFS-7707 as below and in table 2.

$$\begin{aligned}\text{Score (edit log corruption)} &= \text{score (edit)} + \text{score (log)} \\ &\quad + \text{score (corruption)} \\ &= 2.4 + 2.4 + 3 = 7.8 \quad (2)\\ \text{Score (edit log)} &= \text{score (edit)} + \text{score (log)} \\ &= 2.4 + 2.4 = 4.8 \quad (3)\end{aligned}$$

Frequency(edit) is higher than frequency(corruption), degree(edit) is higher than degree(corruption); but ratio i.e. deg(corruption)/freq(corruption) is higher than deg(edit)/freq(edit).

Thus, words that occur predominantly and longer candidate keywords are selected by rapid automatic keyword extraction method. In comparison to other existing keyword extraction methods such as Tf-idf, TextRank, Ngram with tag and others, RAKE achieves higher precision and recall [30], [31].

C. FUZZY CLUSTERING

Clustering is an unsupervised machine learning method which divides the data into distinct clusters. Clusters are formed based on distances or similarity between each data point. The two main approaches of clustering are: hard clustering and soft (fuzzy) clustering. Every data point can belong to only one cluster in hard clustering, i.e. it has value of either 0 or 1.

In soft clustering, every data point can belong to every cluster with a certain degree of membership value. Fuzzy C-means clustering [32] is the most prominent technique among fuzzy clustering techniques. The main objective of fuzzy c-means (FCM) is to minimize the overall distance (Euclidean distance) between data point and center of a cluster. FCM initially selects cluster centers randomly and assign random membership value to every data point for each cluster. After every iteration, it updates the center of the clusters and degree of membership is found by equation (4) and (5).

$$M_{ij} = 1 \Big/ \sum_{K=1}^{C} (E_{ij}/E_{ik})^{(\frac{2}{f}-1)} \quad (4)$$

$$C_j = (\sum\nolimits_{i=1}^{n} \frac{(M_{ij})^f x_i)}{(M_{ij})^f} \quad (5)$$

for all, j= 1, 2, …. C Where, n = number of data points,
C_j = J^{th} cluster center,
f = fuzziness index, f $\mathcal{E}$ (1, infinity)
c= number of cluster center
M_{ij} = membership of i^{th} data to j^{th} cluster center
E_{ij} = Euclidean distance between i^{th} data to j^{th} cluster center

The number of iterations minimizes the objective function i.e. minimizes the Euclidean distance between the i^{th} data and j^{th} cluster center.

$$O_f = \sum\nolimits_{i=1}^{n} \sum\nolimits_{j=1}^{c} (M_{ij})^f \, | \, |x_i - c_j| \, |^2 \quad (6)$$

O_f = Objective function
$| \, |x_i - c_j| \, |^2$ represents Euclidean distance between i^{th} data to j^{th} cluster center [32]–[35].

D. HIERARCHICAL CLUSTERING

Hierarchical clustering [36] is an unsupervised clustering technique that groups similar data into clusters. It is of two types: Agglomerative and Divisive. Agglomerative is bottom up clustering approach which assign each data to its cluster whereas divisive is top down clustering approach, which partitions a single cluster to number of similar clusters. In the proposed approach, agglomerative hierarchical clustering is used. In this, each data point is considered as a separate cluster and then proximity distance of each data point is calculated. Based on proximity, similar clusters are merged together and a single cluster is formed. Iteratively, proximity of newly formed clusters is computed and merged till a single cluster is formed. The result of hierarchical clustering is visualized using a dendogram which is a tree-like structure

that records several sequences of merges. Further, to compute the proximity or similarity between two clusters, several approaches are used. in this work, average linking method is used. In this average distance is calculated between each point in one cluster to each point in another cluster. It is calculated using equation as Where, $D_i \mathcal{E}$ C1 and $D_j \mathcal{E}$ C2 [36]

$$Sim\,(C1, C2) = \sum_{i=1}^{n} \sum_{j=1}^{n} Sim(D_i, D_j)/|C1|^*|C2|^* \quad (7)$$

IV. RESEARCH METHODOLOGY

Various concepts used in this research work are explained in this section. The framework and its various modules are explained in next section followed by the pseudocode of the proposed approach in section IV (E).

The proposed approach focusses on four main concepts of summarization:

- Main content coverage through information richness.
- Diversification of information with minimum content similarity.
- Re-ranking of summary starting with significant and meaningful content.
- Compression ration with respect to original content.

Fig. 1 depicts the framework of proposed methodology to generate extractive summary of software bug reports. Various components are explained as follows:

A. CORPUS CREATION AND TEXT PRE-PROCESSING

A corpus of bug reports of Apache projects (APBRC) is established which consists of twenty-one bug reports. The one-line description, long description and comments of bug reports are extracted and are segmented into sentences and corpus is constructed. The sentences are then pre-processed using standard pre-processing steps: Tokenization, removal of stop words and stemming. After pre-processing, document term matrix (DTM) is constructed. The process is implemented in R language using 'tm' and 'NLP' packages.

B. FEATURE EXTRACTION

After pre-processing of textual data, features are extracted. In feature extraction process, all text features are extracted. All text features are categorized as word level features or sentence level features. Several experiments have been performed with different combination to extract text features to attain best results in terms of relevancy and coverage of bug reports. The various features used in proposed approach are explained.

1) KEYWORD FEATURES

To generate extractive summary, sentences are extracted. Sentence is as collection of words and therefore, to select important sentences, extraction of significant words/keywords is necessary. Two different methods used for keyword extraction are:

1.1 Term-frequency Inverse Document frequency (Tf-Idf) It calculates the term frequency of each word and the inverse document frequency for each word in a document is calculated.

1.2 Rapid Automatic Extraction Process (RAKE)

To automatically extract keywords from documents, RAKE is used. it uses a list of stop words and phrase delimiters. It extracts most relevant keywords or keyword phrases form text by analyzing the frequency of word and its co-occurrences with other words. It assigns a score to each content word (informative word) in a keyword phrase and sum of the score of each content word is the score of keyword phrase. The procedure of assigning a score is illustrated in section III (B).

2) SENTENCE LEVEL FEATURES

After selecting sentences based on most relevant keywords, other features are also investigated which are termed as sentence features. Various sentence level features are described as follows.

2.1 Position of a sentence: It states that leading sentences in document are always important and must be included in a summary. It is computed as:

$$P\,(S_i) = 1 - (^{i\,-\,1}/_{N}) \quad (8)$$

S_i– i^{th} sentence in a document, N- total number of sentences

2.2 Sentence Length: It states that longest sentences are more informative and crucial and must be a part of a summary as compared to short sentences. It is computed as:

$$(S_i) = \frac{number\ of\ words\ in\ S_i}{Total\ numnber\ of\ words\ in\ longest\ sentence} \quad (9)$$

2.3 FUZZY C-MEANS

To extract sentences, Fuzzy C-means algorithm is used. The process of selection of sentences is illustrated as below.

- Firstly, clustering is performed on a set of sentences.
- To apply clustering, computation of optimum number of clusters is necessary. In this approach, four methods are used, Gap Statistics (GSS), K-means, With-in sum of squares (WSS) and Silhouette. The output of all these methods are recorded and same number of clusters computed by two or more methods is selected as optimum number of clusters [37].
- Based on optimum number of clusters, fuzzy C-means clustering is applied. It assigns a degree of membership (probability) to each sentence of belonging to each cluster.
- From each cluster, sentences which are more nearer to the centroid or having high degree of membership (DOM) are selected as they contain more significant information to be included in a summary.
- From each cluster, minimum two and maximum of four sentences are selected, in order to obtain summary with complete coverage and relevancy.

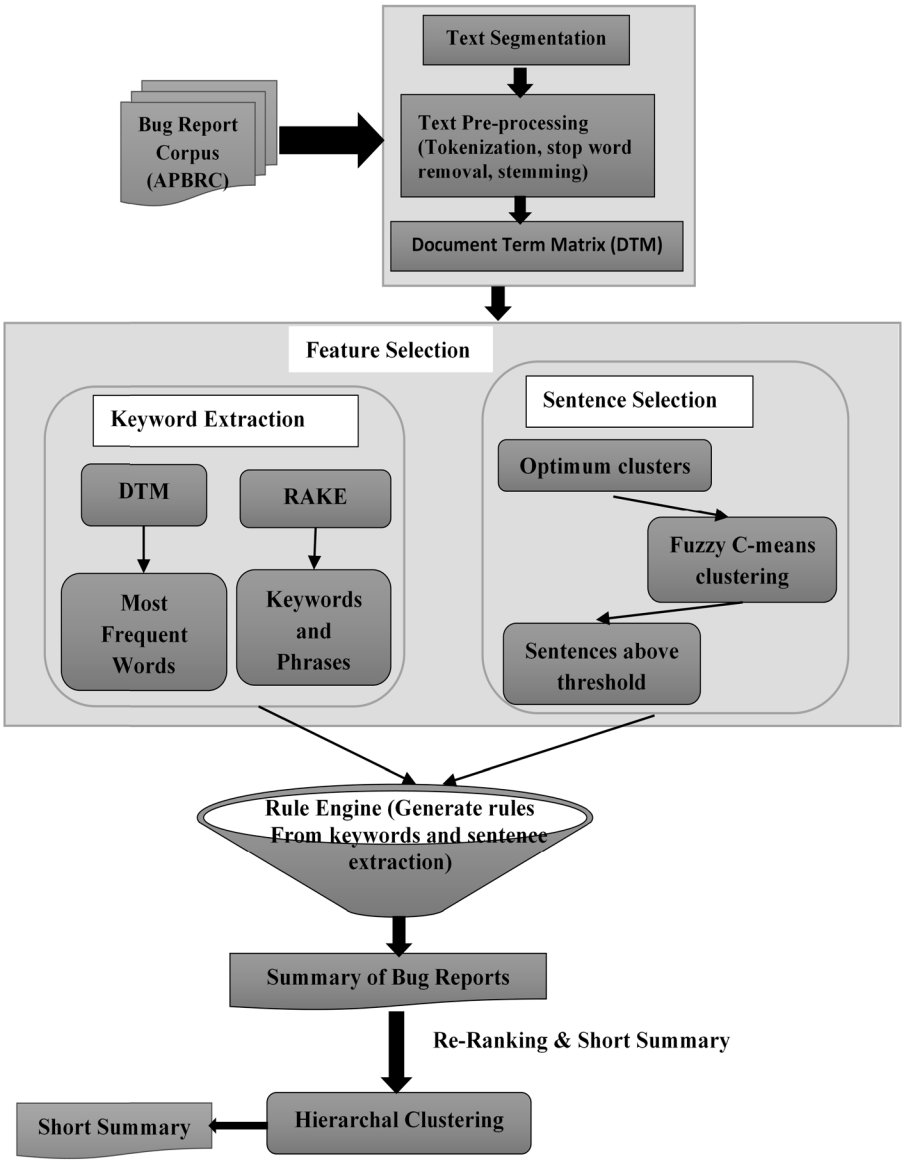


FIGURE 1. Process Flow Diagram of proposed approach.

C. RULE ENGINE

To generate a complete, concise and coherent summary, the process of rule designing is an utmost important phase. All the rules have been built manually with expert domain knowledge. All rules are designed by assigning priority to all the features. Some notations used are:

- Degree of Membership – D.O.M
- Threshold Value – Θ
- Length of a sentence – $L(S_i)$

The designed rules are as follows:

- If (D.O.M $>$ Θ) for each cluster THEN (sentence is selected)
- If (Keyword(score) ≥ 6) THEN (sentence is selected)
- If (Keyword(score)$\geq$1) for two or more keywords THEN (sentence is selected)
- If (BugId is present) $\cap$ (function() is present) THEN (sentence is selected)
- Id ($L(S_i) \geq \Theta$) $\cup$ (Keyword(score) ≥ 1) THEN (sentence is selected)

Based on these above rules, sentences are selected to form an extractive summary. The sentences in a summary are extracted as per their original position in the source document. A compression ratio of 20% with respect to original bug report is achieved.

D. HIERARCHAL CLUSTERING

The generated extractive summary of bug reports is further processed to reduce redundancy of sentences and re-rank the sentences in the summary. To perform re-ranking and redundancy removal, hierarchical clustering is applied to the

generated summary. In average linking method is applied and clusters are visualized as dendograms. Dendograms are tree-like graph structure which depicts hierarchical relationship between sentences. The significance of sentences can be determined by focusing on the height at which they occur in a tree-like structure. The sentences with less height are more significant and are arranged as top sentences. Sentences which are at the same height depicts more similar sentences and thus redundancy can be removed by selecting only one sentence among them.

Thus, after re-arranging the summary sentences on the basis of hierarchical clustering. It was deduced that redundant sentences from the summary are removed. Sentences are re-ranked which depicts that sentences at a lower position in original extractive summary are more important and relevant as per the original document. Thus, hierarchical clustering generates more cohesive, concise and relevant summary. This summary can be considered as short summary which will help developers and other contributors to get a crisp information of original bug report.

The proposed methodology is illustrated through one of bug report of Apache project HDFS-13112 in section V (E).

E. PSEUDO CODE

See Algotithm 1.

V. IMPLEMENTATION

A. DATA COLLECTION

To assess the effectiveness of proposed approach, the prime requirement is a corpus of bug reports. For this, a corpus named Apache Projects Bug Report Corpus (APBRC) is constructed which consists of 21 bug reports of five different projects of Apache Software Foundation: Hadoop-Hdfs, Hadoop-common, hive, groovy and hbase. The bug reports were extracted from the duration of 2015-2018 by a tool named bug report collection system (BRCS). A pool of bug reports was extracted which has immense data such as title, description, comments and other attributes. From this, 21 bug reports were fetched to form APBRC. Bug reports of varied lengths were fetched which consists of minimum 10 number of comments. For construction of corpus, links of patches were removed. The statistical information related to bug reports is presented in table 3. The proposed approach is also evaluated on existing bug report corpus (BRC) which consists of bug reports form four open source projects: mozilla, eclipse, gnome and Kde. All the bug reports and source code is available at link.[1]

Further, newly created bug report corpus APBRC is different from existing BRC corpus in a way that, code snippets are included in bug report while generating the summary, which is not present in bug reports of BRC corpus.

Also, the number of sentences in a bug Report are more in the APBRC corpus as compared to BRC corpus.

[1] https://drive.google.com/drive/folders/1xV2QWHlTtgIauTD1LxTJhVyc0LNHLN4E?usp=sharing

Algorithm 1 Keyword and Sentence Extraction algorithm

Input: S, set of all sentences of bug reports
Output: F_S, Final Summary
for each sentence, $S_i \in S$
Extract bug reports of Apache Projects from Jira Repository
Text Corpus S = Description and comments of bug reports
Decompose the given set S into sentences $S_1, S_2, \ldots\ldots\ldots\ldots S_i$.
Pre-process the corpus S and remove stop words etc..
Create Document term matrix, DTM_i
//Feature Extraction process
// Identify the set of most frequent words
$MFW_i = \emptyset$
for each term, T_r in DTM_i
{
If (frequency of $T_r > 10$)
Then $MFW_i = MFW_i \cup \{T_r$
}
// Automatic keyword extraction
$CW \leftarrow$ set of candidate words
$SW \leftarrow$ set of stop words
for each $CW_i \in CW$
{
$CW_i \leftarrow S_i \cap SW_i$
$Score\,(CW_i) = \frac{Degree(CW_i)}{Frequecy(CW_i)}$
$Score\,(CW) = \sum_{i=1}^{n} CW_i$
}
//For sentence extraction
//To compute optimum number of clusters,N_C
$N_{Ci} <--$ *nbclust* (*Kmeans*)
$N_{cj} <--$ *nbclust* (*GSS*)
$N_{Ck} <--$ *nbclust* (*WSS*)
$N_{ct} <--$ *nbclust* (*Silhouette*)
$N_c <-- \min(N_{ci}, N_{cj}, N_{ck}, N_{ct})$
DOM= Degree of membership
for each $S_i \in S$
{
Compute DOM $\leftarrow fcm(N_i, ''correlation'')$
for each N_c
{
for each S_i
{
if $(DOM \geq \theta)$
$F_S \leftarrow S_i$
}
}
} **//Rule based system**
Rule Generation
for each $S_i \in S$
{
If $(DOM \geq \theta)$
$F_S \leftarrow S_i$
Elseif $(Score\,(CW_i) \geq 6)$ Then
$F_S \leftarrow S_i \supset CW_i$
Elseif $(Score(CW_i \subset \text{CW}) \geq 1)$ Then
$F_S \leftarrow S_i$
Elseif $(S_{LEN} \geq \theta \cup score\,(CW_i) \geq 1)$ Then
$F_S \leftarrow S_i$
}
$F_S \leftarrow \{S_i$
Return F_S
Hierarchal Clustering
Input: Final Summary,F_S
Output: Short Summary,S_S
For each $S_i \in F_S$
{
1) PSEUDO CODE
2) Begin with disjoint clustering with level L(0)= 0 & sequence number, n=0.
3) Let c & r are pair of clusters, $d\,[(c), (r)] = avgd\,[(i), (j)]$
4) $n = n + 1$
Merge cluster (c) and (r) into a single cluster to reach next cluster n $L\,(n) = d\,[(c), (r)]$
1) Update Distance Matrix, D
2) Goto Step (2) till one cluster is reached.

TABLE 3. Statistical information of corpus APBRC.

Name of the Project	Total # of bug reports	# of bug reports selected	Total # of sentences in selected bug reports
Hadoop-Hdfs	10423	3	349
Hadoop-Common	13229	5	620
Hive	13949	5	309
Groovy	7957	2	201
Hbase	16757	6	707

B. ANNOTATION OF BUG REPORTS

Annotation of bug reports by annotators is necessary to create a manual summary. To annotate the bug reports of APBRC corpus, each bug report is assigned to three annotators. Each annotator selects sentences, results of all three annotators are combined and each sentence is assigned a score from 0 to 3. If a sentence is not selected by any annotator, score of 0 is assigned, score 1 is assigned if selected by one annotator, similarly, score 2 and 3 are assigned. Golden standard summary is generated with sentences of scores 2 and 3.

C. RESEARCH QUESTIONS

To analyze the performance of proposed approach for bug summarization, few Research Questions (RQ) have been formulated.

RQ1. Does the proposed approach attain improved results as compared to state-of-the-art approach on existing corpus?

RQ2. Does the proposed approach perform well on other bug reports corpus?

D. EVALUATION METRICS

As per literature, various metrics are used to assess the performance of algorithms viz., Precision, Recall, F-score and Pyramid Precision. **Precision** selects the number of sentences common in both Generated Summary (GS) and Golden Standard Summary (GSS) divided by the total number of sentences in Generated Summary (GS).

$$\text{Precision} = \frac{|\boldsymbol{GS} \cap \boldsymbol{GSS}|}{|\boldsymbol{GS}|} \tag{10}$$

Recall emphasizes on selecting unique number of sentences from Generated as well as Golden Standard summary divided by total number of sentences present in golden standard summary.

$$\text{Recall} = \frac{|\boldsymbol{GS} \cap \boldsymbol{GSS}|}{|\boldsymbol{GSS}|} \tag{11}$$

F-score indicates the harmonic mean of precision and recall calculated and depicts the overall performance of the proposed approach.

$$\text{F-score} = 2.\frac{Precision.Recall}{Precision + Recall} \tag{12}$$

TABLE 4. Extracted keyword phrases with score.

Keyword/Keyword phrases	Score
general {{fseditlog}} thread safety issue	16.366667
normal edit logging activities	11.05113
nointerruptslock technically isn't	9.000
log queue overflows	8.363636
start segment edits	8.18750
expiry isnt essential	8.00000
public void logupdatemasterkey	8.0000
concurrent edit logging	7.55113
abstract secret manager	7.47619
async edit logging	7.05113
token expiration edits	6.06250
locks interruptible method	5.082707
whitespace issues	4.20000
exit condition	4.0000
log corruption	3.8636

Pyramid Precision predicts the annotators perspective. It is measured by the number of links (AL) a sentence is assigned by all the annotators in ratio with maximum number of links assigned to all sentences by annotators.

$$\text{Pyramid precision} = \frac{\#\boldsymbol{AL}(\boldsymbol{topranked sentences})}{\boldsymbol{Total\#AL}(\boldsymbol{Summary length}} \tag{13}$$

E. PROCESS ILLUSTRATION

The process of proposed approach is illustrated through a bug report #Hadoop-Hdfs-13112 from Hadoop-hdfs apache project.

Step 1: Extraction of keywords

For keyword extraction, Pre-processing of textual data is performed and a document term matrix (DTM) is constructed and most frequent terms are extracted.

Cause, Corruption, deadlock, expiration, token, acquire, change, nointerruptslock, transitions, fseditlog, concurrent, getdelegationtoken, locking, interrupte, expose, exception

Step 2: Extraction of Keyword Phrases

This is the second step for keyword extraction. The entire bug report is passes through Rapid automatic keyword extraction module for extracting keyword phrases which are more significant and convey important meaning. The package ''rapidraker'' is installed and keyword phrases with their score is determined. Some of the phrases with high scores are depicted in table 4.

Step 3: Find optimum number of clusters

To select sentences, this is the first step to find optimum number of clusters. Clusters are computed through four different methods: Gap Statistics (GSS), K-means, With-in sum of squares (WSS) and Silhouette. The graphs obtained are depicted in fig.2.

From the obtained graphs, K-means and Silhouette attains optimum number of clusters as 3, while GSS and WSS attains 1 and 7 optimum clusters. Therefore, as two methods

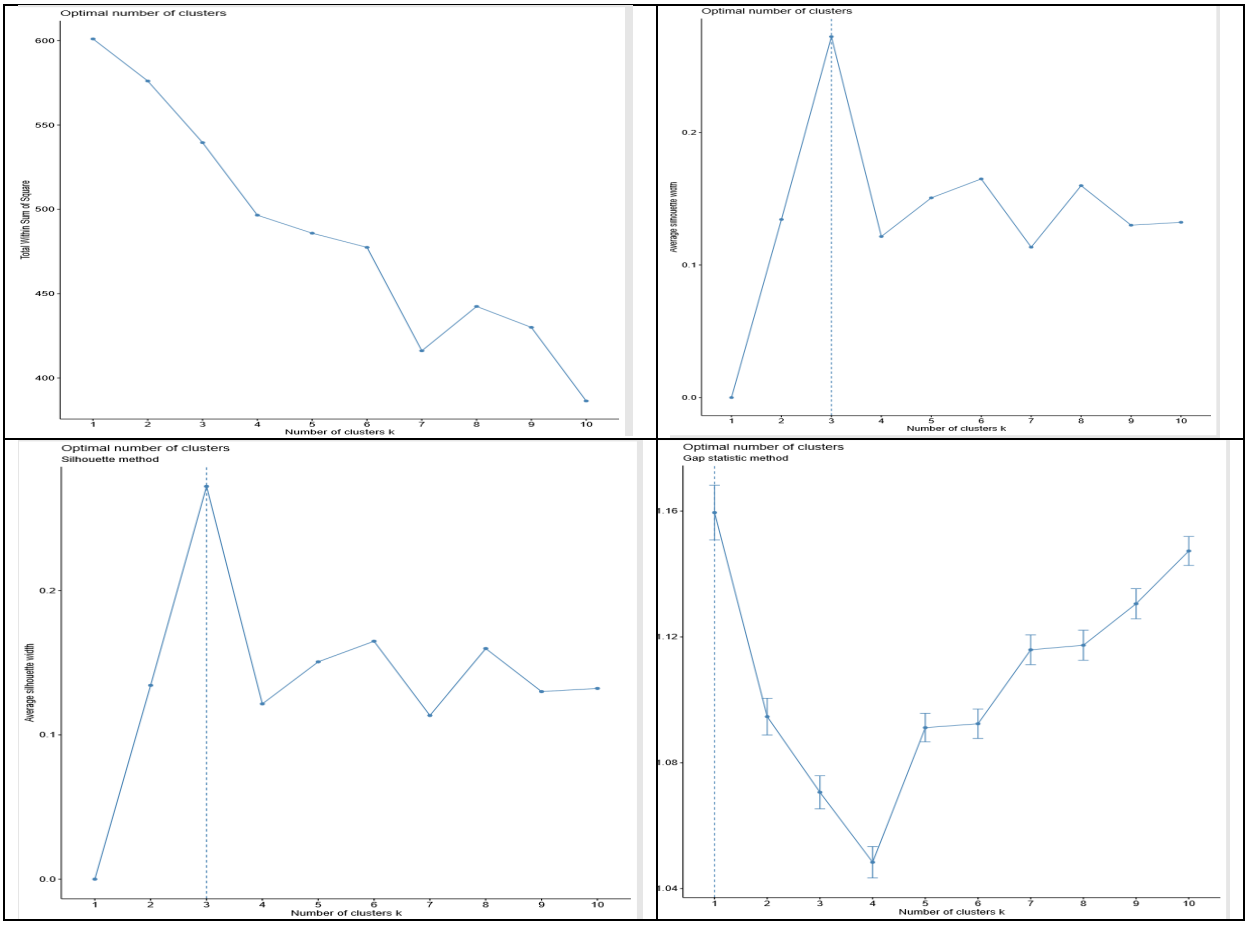


FIGURE 2. Computation of Optimum number of clusters.

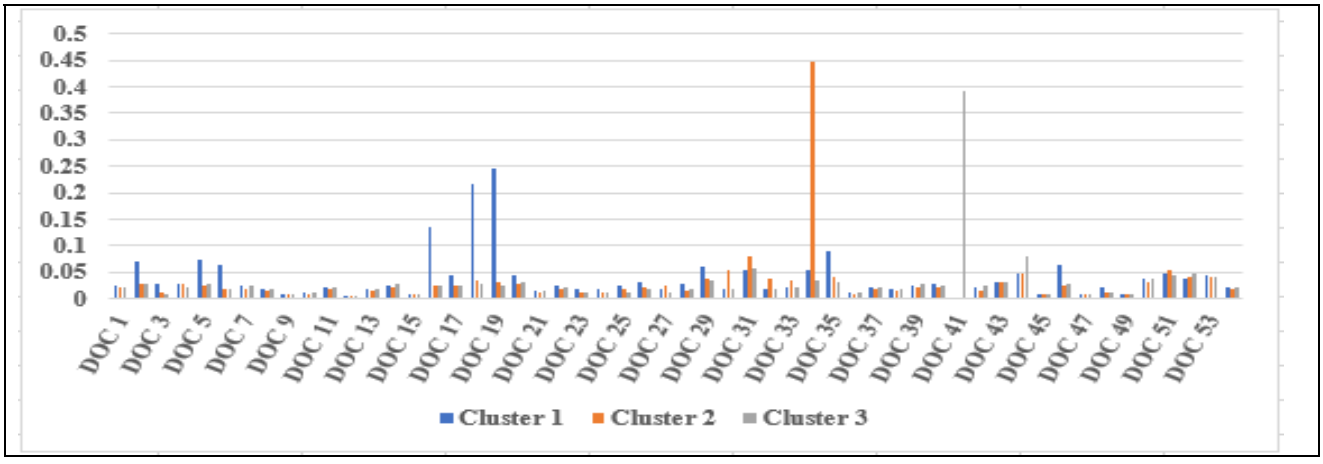


FIGURE 3. Membership value of each sentence.

attains 3 optimum clusters, the dataset is partitioned into three clusters as depicted in fig. 3.

Step 4 **Fuzzy C-Means Clustering**

After attaining optimum number of clusters, fuzzy c-means clustering is applied. In this, membership value is assigned to each sentence using ''correlation'' as fuzzy membership function. Each sentence is assigned a membership value for each cluster. The sentences with high membership value belong to that particular cluster and that sentence is selected as a part of summary. To illustrate this, membership value

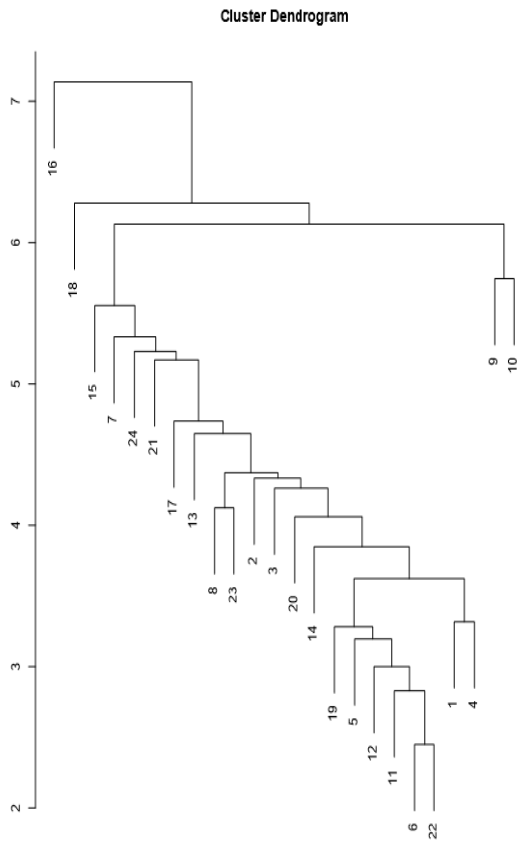
Hierarchal Clustering	Top 5 Sentences
	[16] "So why I do I leave the interrupt set instead of throwing? Less risky to avoid changing the abstract secret manager. Rolling a key will catch the interrupt. If it also decides to expire tokens in the same cycle, it will try to acquire the read lock again, and deadlock again. Leaving the thread interrupted prevents that and allows the run loop to hit the exit condition." [18] "One question\n {code:title=FSNamesystem.java}; public void logUpdateMasterKey(DelegationKey key) { assert hasReadLock(); getEditLog().logUpdateMasterKey(key); getEditLog().logSync(); }{code}" [15] "The abstract secret managers master key roll currently catches ioes and plows ahead. Expects the while (!done) to exit cleanly.Survives the interrupt. But can cause expiry to crash." [9] "And only the secret manager is edit logging with a read lock and all others are using a write lock, there can be no concurrent edit logging and it covers the general {{FSEditLog}} thread safety issue, not only the issue between logging and rolling." [10] Now, if we believe that it is only unsafe between edit logging and rolling (i.e. normal edit logging activities are thread safe), we could make {{getDelegationToken()}}, {{renewDelegationToken()}} and {{cancelDelegationToken()}} acquire a read lock. And perhaps lease-related calls too. Any thoughts on this?"

FIGURE 4. Dendrogram and a short summary.

of all sentences of each cluster are depicted through a graph in fig.3.

Step 5 **Fuzzy Rule Generation**

After sentences are selected from each cluster based on high degree of membership value, other sentences are selected based on other features such as keywords, keyword phrases, sentence length and sentence position. Sentences are examined through various generated rules in section III (B) and if sentence covers that rules, it is selected to be a part of summary.

Step 6 **Generated Summary**

On the basis of several rules generated, final summary is formed. The generated summary as well as manual summary id depicted in table 5.

Step 7: Hierarchical Clustering

The generated summary consists of distinct sentences that can be similar in meaning. Hierarchical clustering is used to remove redundancy and re rank the sentences in generated summary. Redundancy removes the sentences with similar meaning and re-ranks the sentences as any sentence in lower order can describe the full bug report as compared to sentences in higher order as depicted by dendogram in fig. 4.

VI. RESULTS

The research questions formulated to analyze the performance of the proposed approach are discussed and analyzed in this section. The performance of the proposed approach is evaluated in terms of four performance measures: Precision, Recall, F-score and Pyramid Precision.

A. RESEARCH QUESTION 1

In RQ1, the performance of the proposed approach is investigated against the existing supervised and unsupervised approaches for bug report summarization. The experimental results in terms of four evaluation metrics is presented in table and depicted in the form of bar graphs in fig 5. The result illustrate that proposed Automatic keyword and Sentence based approach attains improved results over BRC [19] and LRCA [21] by 34.3%, 25.77%, 12.77% and 24.23%, 16.83%, 6.88% in terms of Recall, F-score and Pyramid Precision whereas BRC and LRCA achieves better result by 2.19% and 3.52% in terms of precision. However, overall performance is measure by F-score in which 25.77% and 16.83% improvement is achieved. When comparing the proposed approach against unsupervised techniques, proposed approach outperforms Hurried [23] by 7.19%, 26.03%, 20.95% and 15.44%

TABLE 5. Manual and generated summary for bug Id #13112.

Manual Summary	Generated Summary
Token expiration edits may cause log corruption or deadlock	Token expiration edits may cause log corruption or deadlock
HDFS-4477 specifically did not acquire the fsn lock during token cancellation based on the belief that edit logs are thread-safe.	HDFS-4477 specifically did not acquire the fsn lock during token cancellation based on the belief that edit logs are thread-safe.
Failure to externally synchronize on the fsn lock during a roll will cause problems.	For sync edit logging, it may cause corruption by interspersing edits with the end/start segment edits.
For sync edit logging, it may cause corruption by interspersing edits with the end/start segment edits.	Async edit logging may encounter a deadlock if the log queue overflows.
Async edit logging may encounter a deadlock if the log queue overflows.	However, HDFS-13051 lost the race with async edits.
Luckily, losing the race is extremely rare. In ~5 years, we've never encountered it.	Submit is pending HADOOP-15212.
The addition of read locks ensures these edit logging activities do not collide with edit rolling or HA transitions(In addition to the level of safety provided by {{noInterruptsLock}}).	The addition of read locks ensures these edit logging activities do not collide with edit rolling or HA transitions(In addition to the level of safety provided by {{noInterruptsLock}}).
A write lock is not required since these don't change any state other threads are accessing with a read lock.	
And only the secret manager is edit logging with a read lock and all others are using a write lock, there can be no concurrent edit logging and it covers the general {{FSEditLog}} thread safety issue, not only the issue between logging and rolling.	And only the secret manager is edit logging with a read lock and all others are using a write lock, there can be no concurrent edit logging and it covers the general {{FSEditLog}} thread safety issue, not only the issue between logging and rolling.
Now, if we believe that it is only unsafe between edit logging and rolling (i.e. normal edit logging activities are thread safe), we could make	Now, if we believe that it is only unsafe between edit logging and rolling (i.e. normal edit logging activities are thread safe), we could make
(think QJM).	(think QJM).
{code:title=FSNamesystem.java}; public void logUpdateMasterKey(DelegationKey key) { assert hasReadLock(); getEditLog().logUpdateMasterKey(key); getEditLog().logSync(); }	One question\n {code:title=FSNamesystem.java}; public void logUpdateMasterKey(DelegationKey key) { assert hasReadLock(); getEditLog().logUpdateMasterKey(key); getEditLog().logSync(); }
I think {{logSync}} is usually done outside of the FSN lock, why not do the same here?	I think {{logSync}} is usually done outside of the FSN lock, why not do the same here?
Also just to confirm my understanding: the comment in {{logExpireDelegationToken}} says that expiration edits are batched, which is reasonable.	Also just to confirm my understanding: the comment in {{logExpireDelegationToken}} says that expiration edits are batched, which is reasonable.
The only "risk" is issuing tokens with a lost key, which isn't an issue because if the token is synced, its secret was implicitly synced.	In code there is no {{logSync}} called at the end of the {{removeExpiredToken}}, but we dont necessarily have to call it because worst case is we lost
{{getDelegationToken()}}, {{renewDelegationToken()}} and {{cancelDelegationToken()}} acquire a read lock. And perhaps lease-related calls too. Any thoughts on this?	{{getDelegationToken()}}, {{renewDelegationToken()}} and {{cancelDelegationToken()}} acquire a read lock. And perhaps lease-related calls too. Any thoughts on this?
The handler for {{enterSafeMode}} has acquired a write lock.	The handler for {{enterSafeMode}} has acquired a write lock.
The secret manager is stuck waiting for read lock.	The secret manager is stuck waiting for read lock.
The noInterruptsLock technically isn't necessary anymore if caller stopping the secret manager has the write lock, but per comments I left it there for safety.	The noInterruptsLock technically isnt necessary anymore if caller stopping the secret manager has the write lock, but per comments I left it there for safety.
	The methods no longer throw InterruptedIOException, but leave or set the interrupt flag if interrupted. Why?
The abstract secret manager's master key roll currently catches ioes and plows ahead. Expects the while (!done) to exit cleanly. Survives the interrupt. But can cause expiry to crash.	The abstract secret managers master key roll currently catches ioes and plows ahead. Expects the while (!done) to exit cleanly.Survives the interrupt. But can cause expiry to crash.
So why I do I leave the interrupt set instead of throwing? Less risky to avoid changing the abstract secret manager. Rolling a key will catch the interrupt. If it also decides to expire tokens in the same cycle, it will try to acquire the read lock again, and deadlock again. Leaving the thread interrupted prevents that and allows the run loop to hit the exit condition.	So why I do I leave the interrupt set instead of throwing? Less risky to avoid changing the abstract secret manager. Rolling a key will catch the interrupt. If it also decides to expire tokens in the same cycle, it will try to acquire the read lock again, and deadlock again. Leaving the thread interrupted prevents that and allows the run loop to hit the exit condition.
Went ahead and individually lock per-token, in the off case there's a glut of tokens to expire and edit logging is being slow	Went ahead and individually lock per-token, in the off case theres a glut of tokens to expire and edit logging is being slow
	it on failover, and new NN will still remove it in the next interval.
	Expiry edits dont need a sync for the reason you state. Failover will expire them. Unlike an explicit cancel, an expiry isnt essential for consistency.

TABLE 6. Results on APBRC corpus.

Number of Sentences in Generated Summary (GS)	21
Number of Sentences in Golden Standard Summary (GSS)	18
Number of sentences in common (GS∩GSS)	16
Precision $\frac{\lvert GS \cap GSS\rvert}{\lvert GS\rvert}$	= (16/21) = 0.7619
Recall $\frac{\lvert GS \cap GSS\rvert}{\lvert GSS\rvert}$	= (16/18)= 0.8888
F-score 2. $\frac{Precision.Recall}{Precision+Recall}$	= 0.8203
Pyramid Precision $\frac{\#\ AL\ (top\ ranked\ sentences)}{Total\ \#\ AL(Summary\ length}$	= (45/53) = 0.8490

in terms of Precision, Recall, F-score and Pyramid Precision respectively. From the bar graph, it is clearly illustrated that proposed approach attains much more improved results compared to other unsupervised algorithm such as Centroid, Maximum Marginal Relevance (MMR), Grasshopper and DivRank. The above observations established that the new proposed approach consistently outperforms other comparative techniques. Thus, it could provide effective and improved summary for bug reports. Shown in Fig 5.

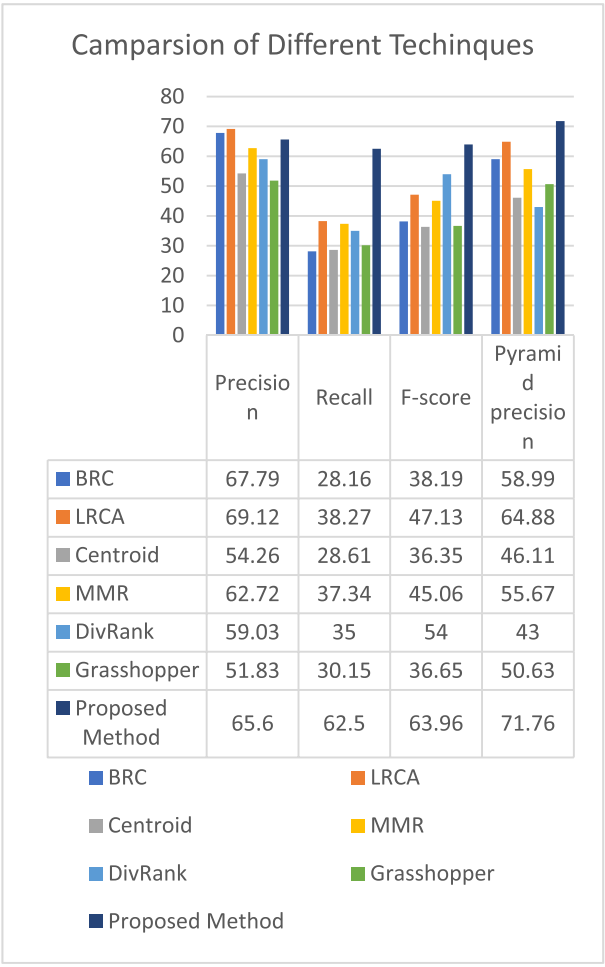


FIGURE 5. Comparison of different algorithm on BRC Corpus.

B. RESEARCH QUESTION 2

To further validate the results of proposed approach, it is also evaluated on newly created APBRC corpus and RQ2 is addressed. The summary of each bug reports is generated based on automatic keyword and sentence extraction using RAKE and Fuzzy C-means clustering. It is an unsupervised learning method and shows an improved result as compared to BRC approach. Average values of 78.22%, 82.18%, 80.10% and 81.66% are obtained for Precision, Recall, F-score and Pyramid Precision respectively. The results achieved in terms of Precision, Recall, F-score and Pyramid Precision are calculated. For illustration #HDFS-13112 as in table 6. Table 7 illustrates the number of sentences in Generated and Golden Standard Summary for all bug reports of APBRC corpus.

Comparison of results on APBRC (21 bug reports) and BRC (36 bug reports) Corpus using Proposed methodology is depicted in fig. 6.

VII. THREATS TO VALIDITY

In this section, various threats to internal and external validity are discussed.

A. INTERNAL THREATS TO VALIDITY

One of the primary threats to internal validity is splitting the paragraphs into sentences. In this study, sentences are splitted when punctuations like '.', ',' '?' '. . . ' are encountered. As different developers have distinct writing style, it is not possible to provide accurate results. Therefore, to minimize the risk, most of sentence terminators are identified and paragraph is split into sentences. Another threat to validity is annotations done by various annotators. As each bug report is interpreted differently by each annotator based on their understanding. To minimize the risk, three annotators are selected to provide annotation to each bug report. A sentence is selected to be a part of summary, if selected by two or more annotators.

B. EXTERNAL THREATS TO VALIDITY

To employ our proposed technique, a corpus consisting of bug reports is required. It does not require any training set as it is an unsupervised approach. For training only few corpora such as brc is available. So, to generate summary of any bug report, without the availability of manually generated golden standard summary, an unsupervised approach is proposed. This reduces the threat as proposed technique is evaluated on other corpus and results are generalized.

VIII. RELATED WORK

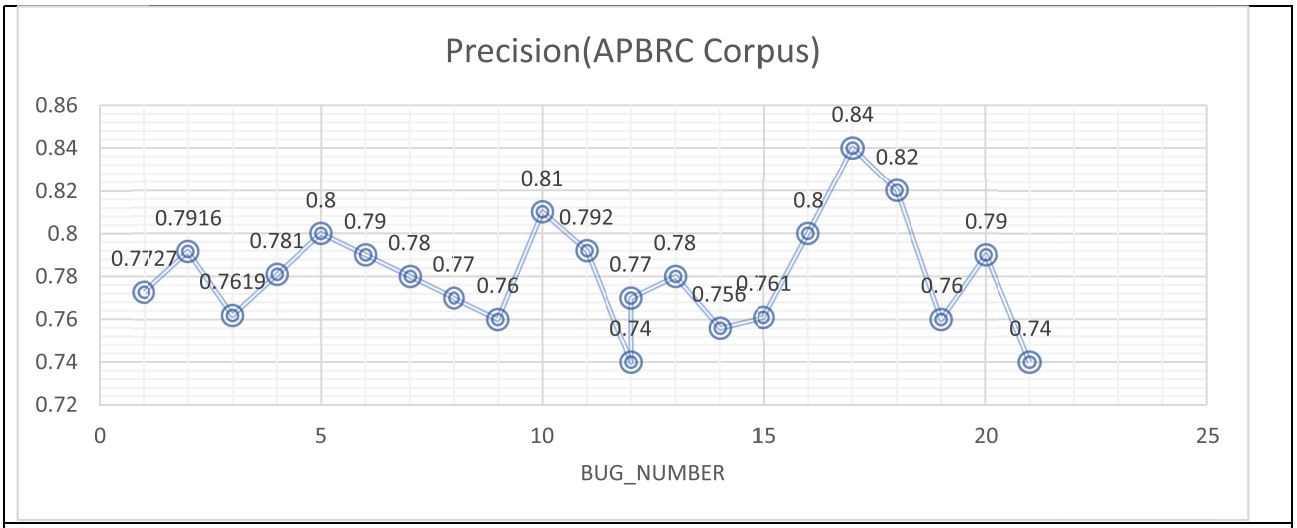
In this section the work done to automatically generate summaries of bug reports summarization approach is explained.

TABLE 7. Detailed description of APBRC corpus.

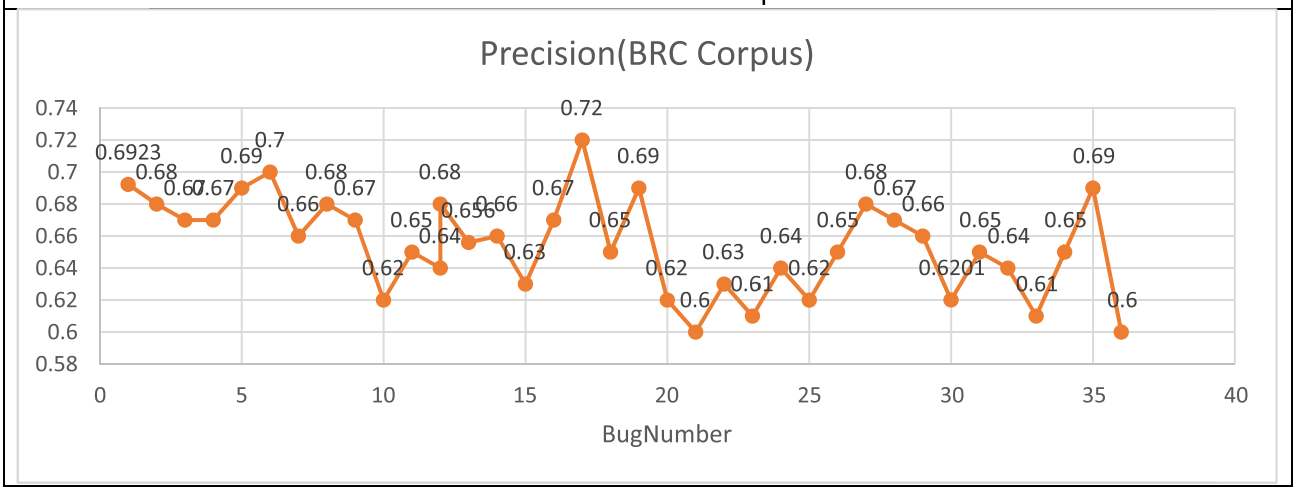
Bug #	# total sentences	# sentences manual summary	#sentences generated summary	Bug #	# total sentences	# sentences manual summary	#sentences generated summary
Bug 1	137	23	22	Bug 12	81	20	19
Bug 2	136	24	24	Bug 13	29	8	7
Bug 3	57	19	21	Bug 14	53	13	11
Bug 4	154	28	26	Bug 15	38	10	9
Bug 5	47	12	10	Bug 16	124	28	29
Bug 6	197	40	41	Bug 17	57	18	19
Bug 7	160	35	34	Bug 18	48	12	11
Bug 8	107	23	23	Bug 19	117	27	25
Bug 9	103	24	23	Bug 20	275	50	52
Bug 10	53	12	13	Bug 21	86	22	21
Bug 11	108	26	27				

A. EXTRACTIVE SUMMARIZATION

Extractive summarization is a method to generate summary of documents by extracting sentences from the original documents. It has been used in several areas of applications such as web articles [2], meeting and telephonic conversations [3], multi-document summarization [1], [38], [39], automatic highlighting of text [40] and many others. To produce an extractive summary various technique are used such as genetic algorithm [41], conditional random fields [42], neural networks [43], semantic similarity such as latent semantic analysis [44]–[46]. Along with these supervised learning techniques, unsupervised learning methods such as fuzzy logic [28], [29], [47], [48], k-means clustering along with term frequency-inverse document frequency [49] has been used. Patel *et al.* proposed a method for multi-document summarization using fuzzy logic. Based on word and sentence features, fuzzy rules were created to generate summary of documents. To remove redundancy among sentences of generated summary, cosine similarity measures is used. The approach was evaluated on DUC 2004 news dataset by several metrics namely, pyramid precision, content coverage score, relative utility and ROUGE [28]. R. Abassi-Ghaletaki *et al.* used fuzzy logic system along with evolutionary algorithm and cellular learning automata to generate a summary of DUC 2002 dataset. The approach was evaluated using ROUGE-1 performance measure and results were compared with several other methods. The results indicate that evolutionary algorithms combined with fuzzy logic method outperforms other techniques [47]. Goularte *et al.* proposed a method that used fuzzy metrics to extract most informative sentences. After pre-processing of textual data, sentence score was identified using several features and scores was normalized in range of 0 to 1. In fuzzy analysis, 27 rules were produced in a rule base system and relevance was computed using bell membership function. The proposed method was evaluated on Portuguese texts collected in VLE. The results conclude that fuzzy summarization improves the informativeness of generated summary along with identification of concepts to guide teachers and students in VLE [29]. In another research, Valdes *et al.* combined semantic graphs and fuzzy logic to generate an extractive summary of documents. A semantic graph is generated between the concepts, based on semantic relationships obtained through wordnet. Further, semantic graphs of each document are merged and concept clustering algorithm is used to identify the relevant topics. For relevance assessment of sentences, fuzzy aggregation function is applied to compute the combined results of several features [48]. Khanna *et al.* [50] proposed an unsupervised extractive summarization approach for summarizing text document. In this, after pre-processing of text data, various features such as bitoken, position of a sentence, tritoken, sentence length, cosine similarity and term frequency-inverse document frequency are extracted. After extraction, features are converted into numerical data and are fed to fuzzifier to attain linguistic value of each feature. Based on obtained values, fuzzy rules are generated and summary is produced. The approach is evaluated on DUC2004 and BBC news datasets using evaluation metrics such as precision, recall and F1 score. Sanchez-Gomez *et al.* [51] performs a comparative study of disparate criterions applicable for generic extractive summarization for multi-documents. In this, authors execute experiments on DUC datasets with all possible combinations of various criterions, namely Content Coverage, Redundancy Reduction, Relevance and Coherence. Authors observed total of eleven possible combinations. Multi-Objective Artificial Bee Colony (MOABC) algorithm has been used as an objective function. Results conclude that including coherence and relevance with content coverage and redundancy reduction attains the best results followed by content coverage, redundancy reduction and relevance. Coherence criteria has the highest computational cost as it requires sorting of sentences. In another work [52], author proposed a novel unsupervised approach Karci Summarization for text data. It measures the extent to which each sentence can represent the meaning of



Precision on BRC Corpus



Recall on APBRC CORPUS

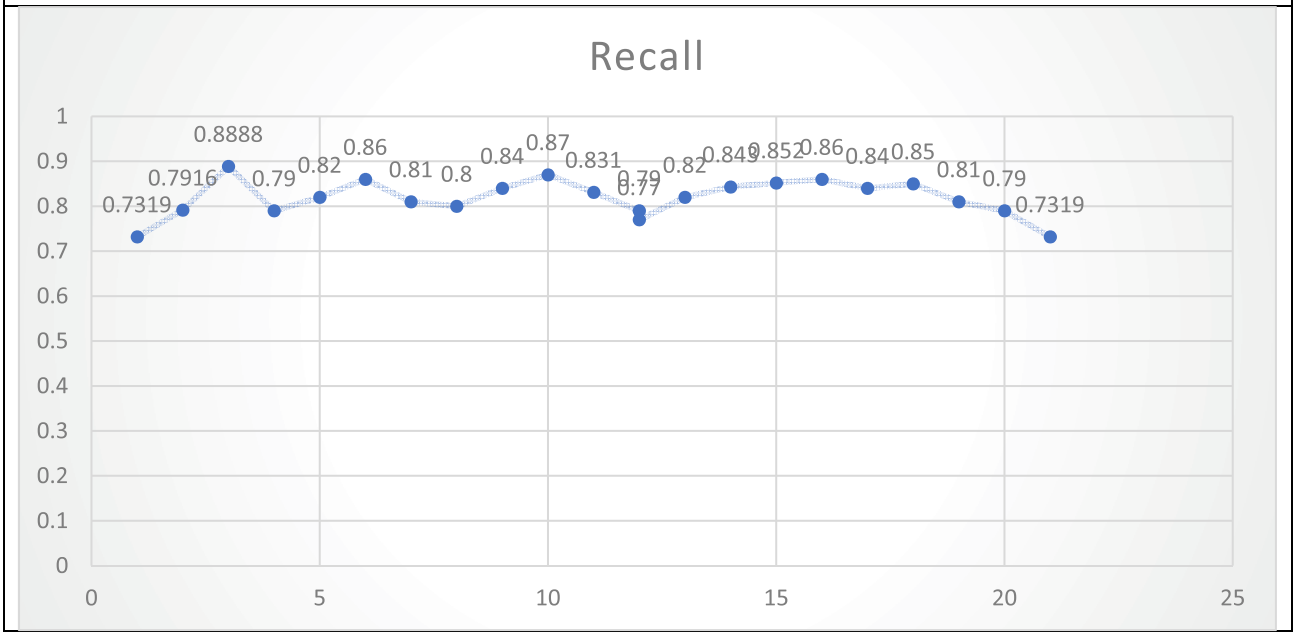


FIGURE 6. Various metrics Precision, Recall, F-Score and Pyramid Precision on APBRC and BRC corpus.

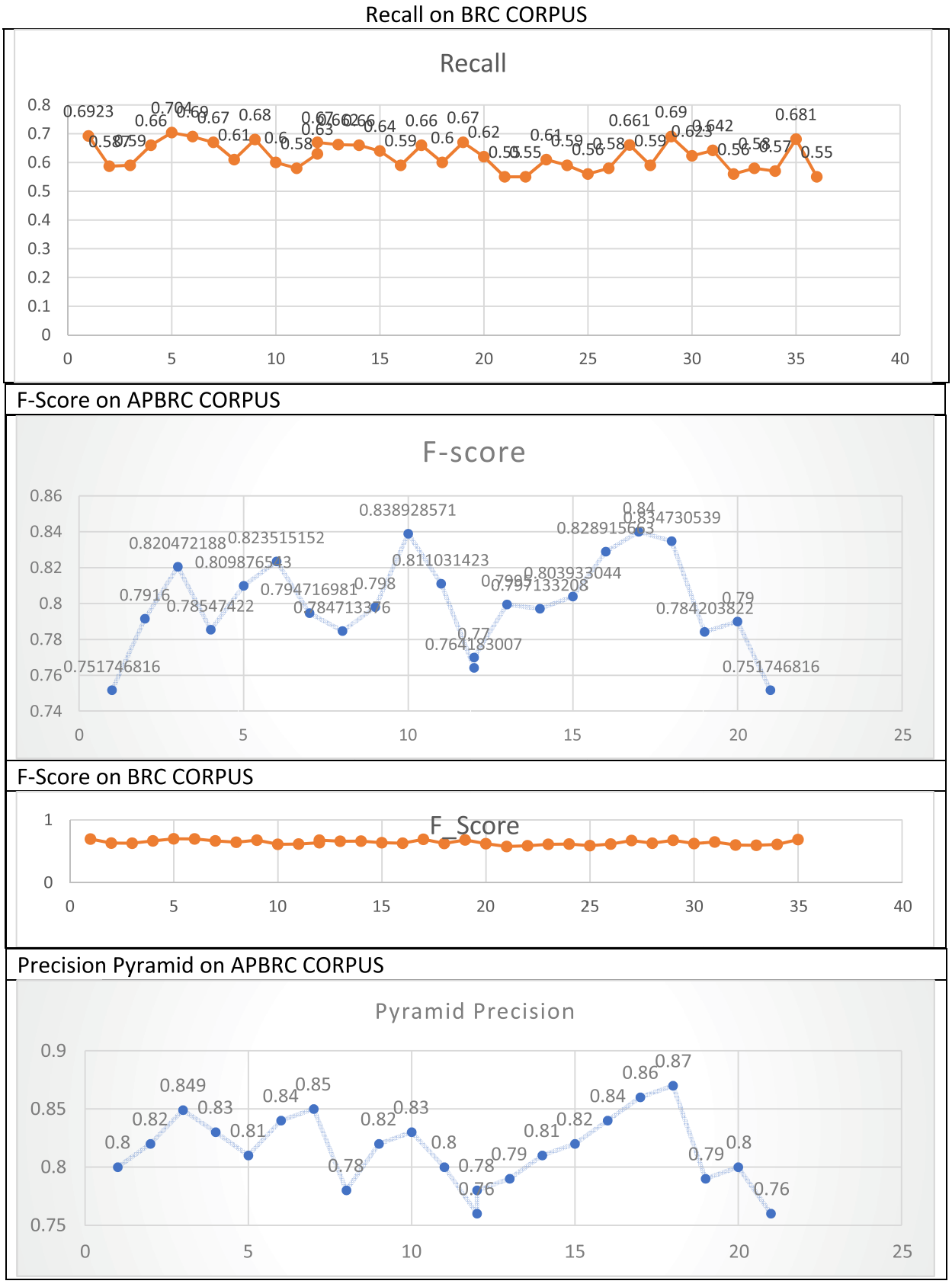


FIGURE 6. *(Continued.)* **Various metrics Precision, Recall, F-Score and Pyramid Precision on APBRC and BRC corpus.**

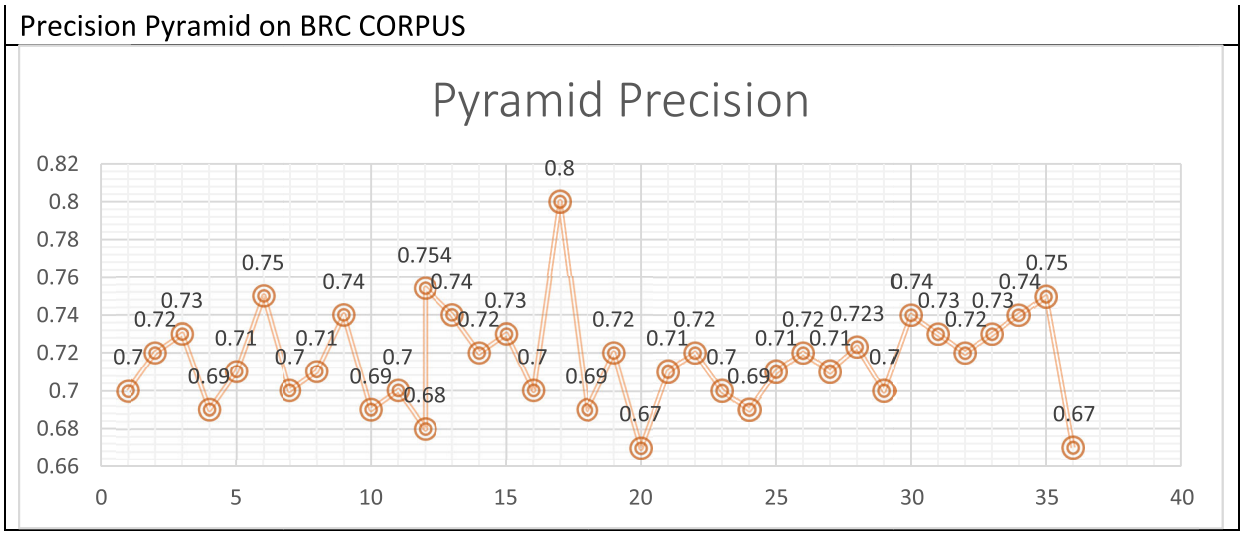


FIGURE 6. *(Continued.)* **Various metrics Precision, Recall, F-Score and Pyramid Precision on APBRC and BRC corpus.**

entire text in a numerical form. In this, for pre-processing of text documents, a tool named KUSH is developed which transfers the relations between sentences into representative graphs. The authors used two significant concepts such as graph theory and entropy. The performance is evaluated in terms of Recall Oriented Understudy for Gisting Evaluation (ROUGE). In successive study [53], author utilizes two concepts: textual graph and maximum independent sets to generate an extractive summary of pre-processed documents. From textual graph, maximum independent sets were identified and nodes with independent sets were removed. The remaining nodes were used as main concepts and are included in the document summary. The approach is evaluated on DUC 2002 and DUC2004 datasets and achieve a rouge score of 0.38072 for 100-word summaries, 0.52954 for 200-word summaries and 0.59208 for 400-word summaries. To address the issue of summarization of multiple documents, Rautray and Chandra [54] proposed a new approach based on Cat Swarm Optimization (CSO). In this approach, after pre-processing of text documents, similarity between sentences is computed and sentences with similarity above threshold value are selected. On selected sentences, CSO algorithm is applied and final summary is generated. The approach is evaluated on DUC data and performance is measures using metrics such as F-score, Rouge Score and Summary Accuracy. The approach is also compared with two other existing approaches namely, Harmony Search and Particle Swarm Optimization. The results indicate that CSO attains improved results as compared to other approaches.

B. BUG SUMMARIZATION

Few researchers have worked on summarization of bug reports that have been reviewed in this section. Rastkar *et al.* used a summarizer developed to summarize e-mail and meeting conversations [4] to summarize bug reports. Thirty-six Bug reports of four open source projects: mozilla, eclipse, Gnome and Kde were extracted to construct a corpus named Bug Report Corpus (BRC). 24 conversational features were computed for each sentence of bug report and probability was computed. The sentences with high probability were selected to be a part of summary. The results were compared with the golden standard summary created by annotators and a precision of 57%, recall of 35%, F-score of 40% and 66% pyramid precision was attained [19]. Jiang *et al.* believed that duplicate bug reports contain information related to master bug reports and can generate more accurate summary. So, he modified existing BRC corpus to add duplicate bug reports of existing master bug reports. Along with 24 conversational features pagerank algorithm was used to compute the textual similarity among sentences. Sentences with high probability value and sentences with high textual similarity were selected and were merged using ranking merger. In contrast to previous study, results were improved in terms of pyramid precision and precision with values of 60.39% and 59.62% respectively [20]. In another work, Ferreira *et al.* proposed that comments of bug reports generate more accurate summaries. Summary of 50 bug reports were generated by ranking the comments using various ranking techniques and conclude that pagerank combined with cosine similarity generates better results [55]. Huai *et al.* proposed a new method of summarization using human intentions. Intentions were classified into seven categories: bug description, fix solution, opinion expressed, seeking information, giving information, meta/code and emotion express. Intentions of BRC corpus were mined and Intention Bug Report Corpus (IBRC) was constructed. The results conclude that improvement was attained for precision, recall, pyramid precision and f-score with values of 5%, 3%, 5% and 3% [56]. Jiang *et al.* [21] proposed a supervised approach to summarize bug reports named as Logistic Regression with Crowdsourcing Attributes

(LRCA). In this proposed approach, a survey is conducted to identify the existing techniques for the process of attribute construction. Further, a new method crowd attribute is proposed to infer attributes from crowd generated data for summarization. By this method, 11 new attributes are constructed under the guidance of heuristic construction rules and LRCA is used to generate more accurate summaries. The approach is evaluated on existing SDS dataset and achieves an improvement of 1.33%, 10.11%, 8.94% and 5.89% for precision, recall, f-score and pyramid precision respectively when compared against state of art BRC approach.

Contrary to these supervised learning methods, few researchers have proposed unsupervised approaches. Mani *et al.* proposed an approach which removes noise from bug reports and classifies bug reports into three types: question, investigative or a code snippet. If a sentence is not classified into any of three categories, it is discarded [22]. Nithya *et al.* used a similar approach to remove duplicated bug reports [57]. Lotufo *et al.* proposed an unsupervised approach to improve the summary generated by email summarizer. Four summarizers were created to test each of the three hypothesis and a combination of all three hypotheses. The approach was evaluated on BRC corpus and an improvement of 12% and 8% was attained in terms of precision and pyramid precision [23]. Kukkar *et al.* proposed an unsupervised approach for bug summarization to generate more informative summary to meet developer's expectation. Four challenges were identified from the previous researches and to overcome these challenges, two features: informative and phrase-ness were extracted to generate all possible subsets for the summary. Particle swarm optimization (PSO) is used to effectively search the semantic text and select the best possible subset of summary. The approach was evaluated on 10 bug reports of BRC corpus [58]. In another work, Li *et al.* [59] incorporates deep neural network to summarize bug reports. The authors proposed an approach, DeepSum, which identifies similar bug reports to form a training set and trained an auto-encoder network to accredit scores to each sentence of new bug report. The approach is evaluated on two datasets: Summary Dataset (SDS) and Authorship Dataset (ADS). The results indicate an improvement of 13.2% and 9.2% in terms of F-score and Rouge-n metrics as compared to state-of-art existing techniques.

In this work, an automated keyword and sentence based unsupervised learning method is proposed. For keyword extraction Tf-Idf and RAKE are employed which extract all significant keywords including code snippets. For Sentence extraction, Fuzzy C-Means algorithm is employed and Rule Engine is developed to incorporate sentences from both keywords and sentence extraction. For summarization, only few attributes are used namely, Tf-Idf, RAKE, Sentence Position, Sentence Length and Fuzzy Clustering as compared to several attributes used in previous work. A final summary of bug report is generated. To further remove redundant sentences in generated summary, Hierarchical clustering is performed and top 5 most significant sentences are extracted to generate a short summary of bug report. Table 8 depicts the comparison of previous studies on bug report summarization.

TABLE 8. Different method in literature.

Author & Year	Learning Method	Algorithm used	Attributes Selected	Corpus
Rastkar et.al. 2014[60]	Supervised	BRC	MXS, MNS, SMS, MXT, MNT, SMT, TLOC, CLOC, SLEN, SLEN2, TPOS1, TPOS2, PPAU, SPAU, COS1, COS2, CENT1, CENT2, PENT, SENT, THISENT, DOM, BEGAUTH, CWS,	BRC
Lotufo et.al., 2012[23]	Un-supervised	Hurried	TITLE, DES, SENTIMENT	BRC
Mani et.al.[61]	Un-supervised	Maximum marginal relevance, centroid grasshopper, divrank	WORD, SENTENCE	BRC
Jiang et.al., 2016 [20]	Supervised	PageRank	MXS, MNS, SMS, MXT, MNT, SMT, TLOC, CLOC, SLEN, SLEN2, TPOS1, TPOS2, PPAU, SPAU, COS1, COS2, CENT1, CENT2, PENT, SENT, THISENT, DOM, BEGAUTH, CWS	BRC, MBRC
Kukkar et.al., 2019 [58]	Unsupervised	Particle Swarm Optimization	INFORMATIVE, PHRASENESS	BRC
Jiang et.al., 2018[21]	Supervised	LRCA	SLOC, SLEN, REP, DES	BRC
Li et.al, 2018 [59]	Unsupervised	DeepSum	WORD, SENTENCE	BRC, ADS

IX. CONCLUSION

This paper proposes an unsupervised approach to automatically summarize software bug reports based on keywords and sentence-based features. To eliminate the drawbacks of corpus-oriented and document-oriented approaches as used in literature, two feature extraction methods are used: Term frequency-Inverse document frequency and Rapid automatic keyword extraction are used. RAKE is language and domain independent and is unsupervised in nature which does not require any domain knowledge. For sentence extraction, bug reports are divided into clusters. To compute optimum number of clusters, four methods: K-means, GSS, Silhouette and WSS are used. For optimum number of clusters obtained, fuzzy c-means clustering is utilized to deal with uncertain information in bug reports and sentences in each cluster with high degree of membership value is selected. Rule based

system is used to combine keyword and sentence features and a cohesive summary is generated. The generated summary may consist of distinct sentences with similar meaning or may contain an informative and most relevant sentence in lower order. Therefore, to reduce redundancy and re-rank the sentences in a generated summary, Hierarchical clustering is performed. Compression ratio of 20% with respect to original bug report is achieved. The approach is proposed for bug report summarization. In bug report summarization since training corpus is not easily available and limited to only BRC corpus this unsupervised learning approach can be applied to any corpus of bug report and will generate relevance and accurate summary. This is due to the fact that fuzzy c-means algorithm reduces the issues of ambiguity, vagueness and incompleteness. Fuzzy C-means clustering extracts the most important and relevant sentences from each cluster, thus providing complete coverage of entire bug report.

The proposed approach is an unsupervised approach which can be evaluated on any dataset to retrieve concise and complete summary as it does not require bug training dataset. The approach is evaluated on newly constructed APBRC corpus and existing BRC corpus. The proposed approach is compared against existing supervised (BRC and LRCA) approaches and Unsupervised (MMR, Centroid, DivRank, Grasshopper and Hurried) approaches. The result illustrate that proposed Automatic keyword and Sentence based approach attains improved results over BRC and LRCA by 34.3%, 25.77%, 12.77% and 24.23%, 16.83%, 6.88% in terms of Recall, F-score and Pyramid Precision whereas BRC and LRCA achieves better result by 2.19% and 3.52% in terms of precision. However, overall performance is measure by F-score in which 25.77% and 16.83% improvement is achieved. Significant improvement had been achieved against existing unsupervised approaches. It also attains an average value for precision, recall, f-score and pyramid precision as 78.22%, 82.18%, 80.1% and 81.66% for APBRC corpus. The proposed approach is an automatic approach and outperforms other existing approaches used for bug summarization.

In future, different clustering algorithms will be employed and their impact will be analyzed. Also, the performance of the proposed approach is evaluated through various performance metrics. However, in literature, ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is used for text documents. It has not been used in evaluating the performance of bug reports. In future, it will be considered to employ on bug reports.

REFERENCES

[1] K. Zechner, "Automatic summarization of open-domain multiparty dialogues in diverse genres," *Comput. Linguistics*, vol. 28, no. 4, pp. 447–485, Dec. 2002.

[2] L. Zhou, E. Hovy, and M. Rey, "A Web-trained extraction summarization system," *Proc. HLT-NAACL Conf.*, May 2003, pp. 205–211.

[3] X. Zhu and G. Penn, "Summarization of spontaneous conversations," in *Proc. 9th Int. Conf. Spoken Lang. Process.*, 2006, pp. 1531–1534.

[4] G. Murray and G. Carenini, "Summarizing spoken and written conversations," in *Proc. Conf. Empirical Methods Natural Lang. Process. EMNLP*, Oct. 2008, pp. 773–782.

[5] O. Rambow and J. Chen, "Summarizing email threads," Tech. Rep., 2004.

[6] S. Wan and K. McKeown, "Generating overview summaries of ongoing email thread discussions," in *Proc. 20th Int. Conf. Comput. Linguistics COLING*, 2004, p. 549.

[7] X. Xia, D. Lo, E. Shihab, and X. Wang, "Automated bug report field reassignment and refinement prediction," *IEEE Trans. Rel.*, vol. 65, no. 3, pp. 1094–1113, Sep. 2016.

[8] A. E. Hassan and T. Xie, "Software intelligence: The future of mining software engineering data," in *Proc. FSE/SDP workshop Future Softw. Eng. Res. FoSER*, 2010, pp. 161–165.

[9] T. Xie, S. Thummalapenta, D. Lo, and C. Liu, "Data mining for software engineering," *Computer*, vol. 42, no. 8, pp. 55–62, Aug. 2009.

[10] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *Proc. 27th IEEE/ACM Int. Conf. Automated Softw. Eng. ASE*, 2012, pp. 70–79.

[11] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng. ICSE*, 2010, pp. 45–54.

[12] H. Mei and L. Zhang, "Can big data bring a breakthrough for software automation?" *Sci. China Inf. Sci.*, vol. 61, no. 5, pp. 1–3, May 2018.

[13] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, "Improving automated bug triaging with specialized topic model," *IEEE Trans. Softw. Eng.*, vol. 43, no. 3, pp. 272–297, Mar. 2017.

[14] T. Zhang, G. Yang, B. Lee, and E. K. Lua, "A novel developer ranking algorithm for automatic bug triage using topic model and developer relations," in *Proc. 21st Asia–Pacific Softw. Eng. Conf.*, Dec. 2014, pp. 246–253.

[15] J. Xuan, H. Jiang, H. Zhang, and Z. Ren, "Developer recommendation on bug commenting: A ranking approach for the developer crowd," *Sci. China Inf. Sci.*, vol. 60, Jul. 2017, Art. no. 072105.

[16] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 379–389.

[17] S. Chopra, M. Auli, and A. M. Rush, "Abstractive sentence summarization with attentive recurrent neural networks," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2016, pp. 93–98.

[18] M. Mohd, R. Jan, and M. Shah, "Text document summarization using word embedding," *Expert Syst. Appl.*, vol. 143, Apr. 2020, Art. no. 112958.

[19] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," *IEEE Trans. Softw. Eng.*, vol. 40, no. 4, pp. 366–380, Apr. 2014.

[20] H. Jiang, N. Nazar, J. Zhang, T. Zhang, and Z. Ren, "PRST: A PageRank-based summarization technique for summarizing bug reports with duplicates," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 27, no. 6, pp. 869–896, Aug. 2017.

[21] H. Jiang, X. Li, Z. Ren, J. Xuan, and Z. Jin, "Toward better summarizing bug reports with crowdsourcing elicited attributes," *IEEE Trans. Rel.*, vol. 68, no. 1, pp. 2–22, Mar. 2019.

[22] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "AUSUM?: Approach for unsupervised bug report summarization," in *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng.*, Nov. 2012, pp. 1–11.

[23] R. Lotufo, Z. Malik, and K. Czarnecki, "Modelling the 'hurried' bug report reading process to summarize bug reports," *Empirical Softw. Eng.*, vol. 20, no. 2, pp. 516–548, Apr. 2015.

[24] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *J. Document.*, vol. 28, no. 1, pp. 11–21, Jan. 1972.

[25] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, Nov. 1975.

[26] D. Engel, "Mining for Emerging Technologies Within Text Streams and Documents," in *Proc. Int. Conf. Data Mining. Soc. Ind. Appl. Math.*, Feb. 2009, pp. 1–18.

[27] S. Rose, D. Engel, N. Cramer, and W. Cowley, "CO RI automatic keyword extraction," *Text Mining, Appl. Theory*, vol. 1, pp. 1–20, 2010.

[28] D. Patel, S. Shah, and H. Chhinkaniwala, "Fuzzy logic based multi document summarization with improved sentence scoring and redundancy removal technique," *Expert Syst. Appl.*, vol. 134, pp. 167–177, Nov. 2019.

[29] F. B. Goularte, S. M. Nassar, R. Fileto, and H. Saggion, "A text summarization method based on fuzzy rules and applicable to automated assessment," *Expert Syst. Appl.*, vol. 115, pp. 264–275, Jan. 2019.

[30] A. Kaur and S. G. Jindal, "Bug report collection system (BRCS)," in *Proc. 7th Int. Conf. Cloud Comput., Data Sci. Eng. Confluence*, Jan. 2017, pp. 697–701.

[31] S. Rose, D. Engel, and N. Cramer, "Automatic keyword extraction from individual documents," *Text Mining, Appl. Theory*, vol. 1, pp. 1–20, Mar. 2010.
[32] F. Lobo, "Fuzzy c-means algorithm Fuzzy c-means algorithm," Tech. Rep.
[33] S. K. Lakshmanaprabu, K. Shankar, D. Gupta, A. Khanna, J. J. P. C. Rodrigues, P. R. Pinheiro, and V. H. C. de Albuquerque, "Ranking analysis for online customer reviews of products using opinion mining with clustering," *Complexity*, vol. 2018, pp. 1–9, Sep. 2018.
[34] A. Karami, A. Gangopadhyay, B. Zhou, and H. Kharrazi, "Fuzzy approach topic discovery in health and medical corpora," *Int. J. Fuzzy Syst.*, vol. 20, no. 4, pp. 1334–1345, Apr. 2018.
[35] N. Statistical, S. Ncss, and A. R. Reserved, "Fuzzy clustering," Tech. Rep.
[36] N. Statistical, S. Ncss, and A. R. Reserved, "Hierarchical clustering /dendrograms,"
[37] C. Malika, N. Ghazzali, V. Boiteau, and A. Niknafs, "NbClust: An R package for determining the relevant number of clusters in a data Set," . *Stat. Softw.*, vol. 61, no. 6, pp. 1–36, 2014.
[38] V. K. Gupta and T. J. Siddiqui, "Multi-document summarization using sentence clustering," in *Proc. 4th Int. Conf. Intell. Human Comput. Interact. (IHCI)*, Dec. 2012, pp. 1–5.
[39] D. Wang, T. Li, S. Zhu, and C. Ding, "Multi-document summarization via sentence-level semantic analysis and symmetric matrix factorization," in *Proc. 31st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. SIGIR*, 2008, p. 307.
[40] F. A. Grootjen and G. E. Kachergis, "Automatic text summarization as a text extraction strategy for effective automated highlighting," 2018.
[41] J. R. Simón, Y. Ledeneva, and R. A. García-Hernández, "Calculating the significance of automatic extractive text summarization using a genetic algorithm," *J. Intell. Fuzzy Syst.*, vol. 35, no. 1, pp. 293–304, Jul. 2018.
[42] D. Shen, J. Sun, H. Li, Q. Yang, and Z. Chen, "Document summarization using conditional random fields," *Science*, vol. 80, vol. 7, pp. 2862–2867, 2004.
[43] A. Sinha, A. Yadav, and A. Gahlot, "Extractive text summarization using neural networks," 2018, *arXiv:1802.10137*. [Online]. Available: https://arxiv.org/abs/1802.10137
[44] C. Yadav and A. Sharan, "A new LSA and entropy-based approach for automatic text document summarization," *Int. J. Semantic Web Inf. Syst.*, vol. 14, no. 4, pp. 1–32, Oct. 2018.
[45] J. Steinberger and K. Ježek, "Using latent semantic analysis in text summarization," in *Proc. ISIM*, Apr. 2004, pp. 93–100.
[46] Y. Gong and X. Liu, "Generic text summarization using relevance measure and latent semantic analysis," in *Proc. 24th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. SIGIR*, 2001, pp. 19–25.
[47] R. Abbasi-ghalehtaki, H. Khotanlou, and M. Esmaeilpour, "Fuzzy evolutionary cellular learning automata model for text summarization," *Swarm Evol. Comput.*, vol. 30, pp. 11–26, Oct. 2016.
[48] E. Valladares-vald, A. Sim, J. A. Olivas, and F. P. Romero, "A fuzzy approach for sentences relevance assessment in multi-document summarization," in *Proc. Int. Workshop Soft Comput. Models Ind. Environ. Appl.*, vol. 1, 2020, pp. 57–67.
[49] R. Khan, Y. Qian, and S. Naeem, "Extraction based approach for text summarization using k-means clustering," *J. Sci. Res. Publications*, vol. 4, pp. 1–4, Nov. 2014.
[50] A. Khanna, D. Gupta, S. Bhattacharyya, V. Snasel, J. Platos, and A. E. Hassanien, *International Conference on Innovative Computing and Communications*, vol. 1. Springer, 2019.
[51] J. M. Sanchez-Gomez, M. A. Vega-Rodríguez, and C. J. Pérez, "Experimental analysis of multiple criteria for extractive multi-document text summarization," *Expert Syst. Appl.*, vol. 140, Feb. 2020, Art. no. 112904.
[52] C. Hark and A. Karcı, "Information processing and management Karcı summarization?: A simple and effective approach for automatic text summarization using Karc ? entropy Karc? summarization?: A simple and effective approach for automatic text summarization using Karcı entropy," *Inf. Process. Manag.*, vol. 57, no. 3, p. 102187, 2020.
[53] T. Uçkan and A. Karcı, "Extractive multi-document text summarization based on graph independent sets," *Egyptian Informat. J.*, 2020.
[54] R. Rautray and R. C. Balabantaray, "Cat swarm optimization based evolutionary framework for multi document summarization," *Phys. A, Stat. Mech. Appl.*, vol. 477, pp. 174–186, Jul. 2017.
[55] I. Ferreira, E. Cirilo, V. Vieira, and F. Mourao, "Bug report summarization: An evaluation of ranking techniques," in *Proc. X Brazilian Symp. Softw. Compon., Architectures Reuse (SBCARS)*, Sep. 2016, pp. 101–110.
[56] B. Huai, W. Li, Q. Wu, and M. Wang, "Mining intentions to improve bug report summarization," in *Proc. 30th Int. Conf. Softw. Eng. Knowl. Eng.*, Jul. 2018, pp. 320–363.
[57] R. Nithya and A. Arunkumar, "Summarization of bug reports using feature extraction," *Int. J. Comput. Sci. Mob. Comput.*, vol. 52, no. 2, pp. 268–273, 2016.
[58] A. Kukkar, R. Mohana, A. Nayyar, J. Kim, B.-G. Kang, and N. Chilamkurti, "A novel deep-learning-based bug severity classification technique using convolutional neural networks and random forest with boosting," *Sensors*, vol. 19, no. 13, p. 2964, 2019.
[59] X. Li, H. Jiang, D. Liu, Z. Ren, and G. Li, "Unsupervised deep bug report summarization," in *Proc. 26th Conf. Program Comprehension ICPC*, 2018, pp. 144–155.
[60] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng. ICSE*, Jan. 2010, p. 505.
[61] S. Mani, S. Nagar, D. Mukherjee, R. Narayanam, V. S. Sinha, and A. A. Nanavati, "Bug resolution catalysts: Identifying essential non-committers from bug repositories," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 193–202.

SHUBHRA GOYAL JINDAL (Student Member, IEEE) received the M.Tech. degree in information technology from Guru Gobind Singh Indraprastha University. She is a Ph.D. Research Scholar with the School of Information and Communication Technology, Guru Gobind Singh Indraprastha University. Her research interests include software engineering, machine learning, and text mining.

ARVINDER KAUR received the M.E. degree in computer science from the Thapar Institute of Engineering and Technology, and the Ph.D. degree from Guru Gobind Singh Indraprastha University. She worked with the Dr. B.R. Ambedkar Regional Engineering College, Jalandhar, from 1993 to 2000 and the Thapar Institute of Engineering and Technology, from 1990 to 1993. She has also worked for Gabriel India Ltd., Parwanoo, as an Engineer (R&D). She is a Professor with the School of Information and Communication Technology, Guru Gobind Singh Indraprastha University. Her research interests include software engineering, object-oriented software engineering, software metrics, microprocessors, operating systems, artificial intelligence, and computer networks. She is also a Lifetime Member of ISTE and CSI.